AIAA

NAG 1-613
IN-61-CR
146720
9P.

AN IMPROVED SIMULATED ANNEALING ALGORITHM FOR STANDARD CELL PLACEMENT

Mark Jones and Prithviraj Banerjee

Computer Systems Group
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1101 W. Springfield Av.
Urbana, IL-61801
(217) 333-6564

Acknowledgment: This research was supported in part by the National Aeronautics and Space Administration under Contract NASA NAG 1-613.

Submitted to International Conference on Computer Design (ICCD '87)
New York, Oct 2-4, 1987.

1. Introduction

Simulated annealing is a general purpose Monte Carlo optimization technique that has been applied extensively in recent years [1, 2, 3, 4]. In VLSI design, this technique has been applied to the problem of placing standard logic cells in a VLSI chip so that the total interconnection wire length is minimized [5, 6, 7]. Recently, many researchers have started to investigate techniques for speeding up simulated annealing algorithms by running them on parallel processor systems [8, 9, 10, 11, 12, 13]. Many of these parallel algorithms have been reported to be considerably faster, and to converge to a final placement which is more optimal than similar uniprocessor simulated annealing algorithms. The better performance of these parallel algorithms appears to be caused by the constraints that they place on the distances over which moves can occur, and the use of slightly outdated cell placement information caused by multiple interacting moves being accepted at each parallel iteration.

In this paper, we present an improved standard cell placement algorithm that takes advantage of the performance enhancements that appear to come from parallelizing the uniprocessor simulated annealing algorithm. An outline of this new algorithm is given in Figure 1. Two important differences between this algorithm and previously reported uniprocessor algorithms are in Steps 6 and 9. First, a modified generate function is proposed in Step 6 to allow for nonuniformly distributed distances over which exchange and displacement moves can take place; this technique will be called *multi-windowing*. Secondly, multiple cell movements are considered before updating cell placement data in Step 9, hence the cost calculations are based on slightly outdated placement data. This technique is referred to as *pseudo-parallel moves*.

2. Use of Multiwindowing

In the parallel versions of the simulated annealing placement algorithm, it appears that the average distance a cell moves in the course of the algorithm has a profound effect on the final placement. Specifically, it appears that movement of cells should be biased so that they are restricted more to their local vicinity. Also, parallel simulated annealing algorithms running on architectures

such as the Hypercube are constrained to certain probability distributions because of the way the cells are mapped to the individual processors, i.e., the movement of cells from one section to certain other sections in the physical circuit space is not possible in a single move because processing nodes controlling those sections of the circuit space are not directly connected. A uniprocessor version of the simulated annealing algorithm is not constrained in this manner and can therefore incorporate rather complex windowing techniques and distance probability distributions. For example, in Figure 2, if cell M was picked to perform a displacement, a simple triple windowing scheme could be used to determine where the cell will be displaced to. In Figure 2, the outermost window (W1) is always equal to the physical work space of the circuit. The inner windows, W2 and W3, have sizes proportional to 2/3 and 1/3 of the physical work space and are centered about cell M. In order to favor local movement, the probability of being in the innermost window (W3) is made greater than being in the outer windows. In Figure 2, cell M has a 50% probability that its proposed new position will be within the innermost window W3, a 25% probability of being within window W2 but not window W3, and a 25% probability of being in the physical work space but not within windows W2 or W3.

3. Use of Pseudoparallel Moves

In Figure 1, a conditional data update statement has been added which allows multiple number of accepted moves to accumulate before new cell locations are updated. This amounts to allowing all moves after the first successful move determine the cost function on the basis of outdated placement information. For example, in Figure 3(a), if module M1 is successful in performing a displacement from (x_1, y_1) to (x_2, y_2) during the first iteration of the inner loop, then the circuit should be as shown in Figure 3(b), but because M1's position is not updated, the remainder of the cells still calculate cost functions which involve M1 as though it were still at position (x_1, y_1). Because of this, if module M2, which is connected to M1 via a net connection, is chosen for an attempted move during iteration two, then the half-perimeter wiring cost associated with the net will be computed using the old position of M1.

After each move acceptance, a counter is incremented to keep track of the number of successful moves, since the last cell position update and the new positions of the cells are placed in temporary storage for use later in updating the cell positions. Random cell selection for movement in subsequent iterations is not able to select cells which have made successful moves, but whose positions have not yet been updated. This amounts to freezing the cells' positions until the required number of moves has been accepted. After a specified number of successful moves, the conditional if statement criterion will be satisfied, and all cell positions will be updated using the information saved in temporary storage.

The effect of using slightly outdated information appears to give a higher probability of getting out of local minima, since this technique will accept a higher percentage of moves with uphill changes in the cost function. The *accept* function limits the magnitude of uphill moves but does not affect the total number. The number of uphill moves is affected by the random cell selection and the "observed" placement. By having the observed placement slightly different from the actual placement, it appears more uphill moves are accepted. By having greater numbers of uphill moves which are all limited in magnitude by the *accept* function, the probability of getting out of local minima is increased.

4. Placement Results

The advantage of this algorithm over conventional, uniprocessor simulated annealing algorithms such as TimberWolf is that it converges to a better final placement in a given amount of time. We have implemented the algorithm in the C programming language on a Gould 9050 computer system, running under UNIX 4.2.

We first report the performance of this algorithm using a small 64-standard cell circuit, which was randomly generated and has several known clusters of cells with high connectivity. At each temperature, approximately 100 new state moves were attempted per cell. The first set of tests was concerned with determining the optimal number of moves that should be accepted before cell updating is performed without any windowing. Table 1 shows final placement costs generally

decreasing as the number of multiple moves is increased. The optimal solution occurs when 16 moves have to be accepted before placement update will occur.

Simple testing of a few windowing schemes using 16 multiple moves showed consistent decreases in final placement cost over using windowing alone as seen in Table 2. Because of this, the remainder of the windowing scheme testing was performed using 16 multiple moves. A few of the more promising windowing schemes were applied to two larger industry standard circuits of sizes 183 and 286 cells with promising results, as shown in Table 3.

A variety of windowing schemes and size of windows have been extensively tested, results of which will be reported in the full paper. Several generalized results can be deduced from these studies. For example, if the inner windows are not significantly smaller than the physical work space, the final placements tend to be decidedly inferior. This is in agreement with an earlier observation that the movement of cells should be localized to the area immediately surrounding the cell. This statement is reinforced by noting that in all the windowing schemes, better performance is generally obtained as the percentage of localized moves is increased.

5. Conclusions

This paper proposed an improved uniprocessor simulated annealing placement algorithm which is based on the results obtained from parallelization of the simulated annealing algorithm proposed earlier by us and other researchers. Two new techniques were incorporated: multi-windowing and pseudo-parallel moves. The algorithm has already been implemented and detailed results on the performance of the algorithm on various industry example circuits will be reported in the full paper. The results show that the algorithm is faster and better (in terms of final placement quality for a fixed number of moves) than conventional algorithms.

REFERENCES

[1] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671-680, May 1983.

[2] S. Nahar, S. Sahni, and E. Shragowitz, "Experiments with Simulated Annealing," *Proc. 22nd Design Automation Conf.*, pp. 748-752, June 1985.

[3] S. R. White, "Concepts of Scale in Simulated Annealing," *Proc. Int. Conf. Computer Design (ICCD84)*, pp. 646-650, Oct. 1984.

[4] J. W. Greene and K. J. Supowit, "Simulated Annealing without Rejected Moves," *Proc. Int. Conf. Computer Design (ICCD84)*, pp. 658-663, Oct. 1984.

[5] C. Sechen and A. S. Vincentelli, "The TimberWolf Placement and Routing Package," *Proc. Custom Integrated Circuits Conf.*, pp. 522-527, May 1984.

[6] C. Sechen and A. S. Vincentelli, "TimberWolf3.2: A New Standard Cell Placement and Global Routing Package," *Proc. 23rd Design Automation Conf.*, pp. 432-439, Jun. 1986.

[7] L. K. Grover, "A New Simulated Annealing Algorithm for Standard Cell Placement," *Proc. Int. Conf. on Computer-Aided Design*, pp. 378-380, Nov. 1986.

[8] E. H. L. Aarts, F. M. J. de Bont, E. H. A. Habers, and P. J. M. van Laarhoven, "Parallel Implementations of the Statistical Cooling Algorithm," *Integration, the VLSI Journal*, vol. 4, pp. 209-238, 1986.

[9] M. J. Chung and K. K. Rao, "Parallel Simulated Annealing for Partitioning and Routing," *Proc. IEEE Int. Conf. on Computer Design (ICCD-86)*, pp. 238-242, Oct. 1986.

[10] S. A. Kravitz and R. A. Rutenbar, "Multiprocesssor-Based Placement by Simulated Annealing," *Proc. 23rd Design Automation Conf.*, pp. 567-573, Jun. 1986.

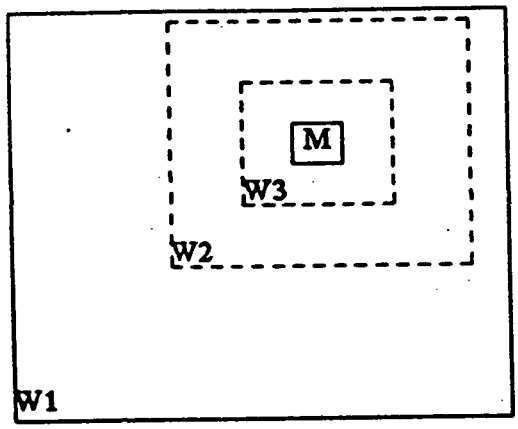
[11] R. A. Rutenbar and S. A. Kravitz, "Layout by Annealing in a Parallel Environment," *Proc. IEEE Int. Conf. on Computer Design (ICCD-86)*, pp. 434-437, Oct. 1986.

[12] P. Banerjee and M. Jones, "A Parallel Simulated Annealing for Standard Cell Placement on a Hypercube Computer," *Proc. IEEE Int. Conf. Computer-Aided Design (ICCAD-86)*, Nov. 1986.

[13] M. Jones and P. Banerjee, "Performance of a Parallel Algorithm for Standard Cell Placement on the Intel Hypercube," *Proc. 24th Design Automation Conf.*, June 1987.

STEP 1. Perform initial random placement of N standard cells
STEP 2. Determine initial temperature.
STEP 3. While "Stopping criteria" : temperature < 0.1 not reached
STEP 4. Generate new temperature
STEP 5. For inner_loop_count = 1 to (N × attempt_parameter)
STEP 6. Generate new move using multi-windowing technique
STEP 7. Evaluate change in cost for move
STEP 8. Accept/reject move using exponential function
STEP 9. IF the number of accepted moves is equal to limit (max_accepted)
THEN update all saved cell positions and zero number of accepted moves counter
ELSE increment accepted moves counter and save cell movements in temporary storage
STEP 10. ENDFOR;
STEP 11. ENDWHILE;

Figure 1. Improved simulated annealing algorithm.



Window	Fraction of Max Dimension	Example Size	Probability Within Window
W1	1	120 X 10	25%
W2	2/3	72 X 60	25%
W3	1/3	36 X 30	50%

Figure 2. Example use of windowing in determining cell movement for cell M.

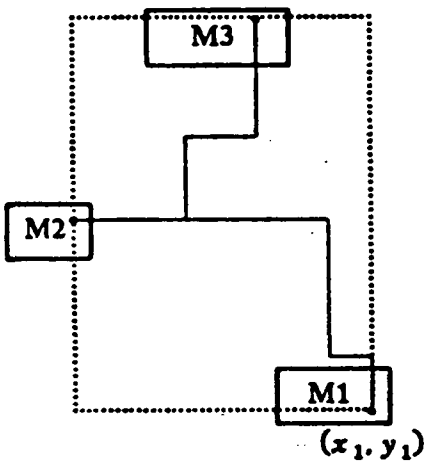


Figure 3(a). Original net placement.

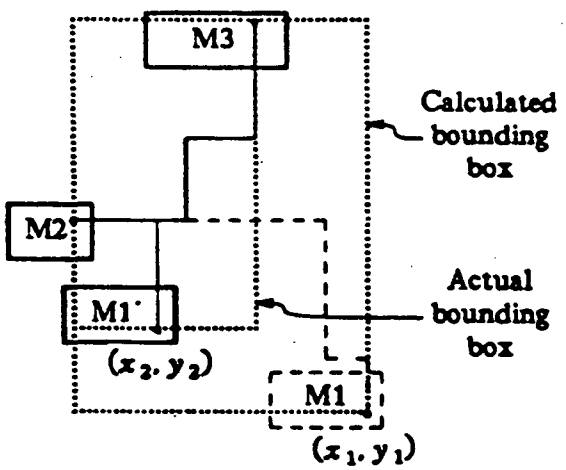


Figure 3(b). Placement after initial acceptance.

Table 1. Cost vs number of multiple moves for 64-cell circuit.

Number of Multiple Moves	Final Placement Cost	Percentage Change
1	24125	+0.0%
2	24138	+0.1%
4	24003	-0.5%
8	23984	-0.6%
12	23924	-0.8%
16	23821	-1.3%
24	24173	+0.2%
32	24829	+2.9%

Table 2. Comparison of final cost for using or not using 16 multiple moves.

Number Windows	Window Sizes as Fraction of max	Distribution of Moves in Windows	Final Cost multiple moves	Percent Change
1	1	100%	23821	-1.3%
2	1 : ¼	20% : 80%	22893	-4.1%
2	1 : ½	13% : 87%	23654	-2.7%
2	1 : ¾	10% : 90%	23823	-1.7%
3	1 : 2/3 : 1/3	10% : 30% : 60%	22148	-7.3%
4	1 : ¾ : ½ : ¼	6% : 12% : 24% : 58%	22813	-3.1%

Table 3. Comparison of cost vs windowing scheme for industry standard circuits.

Number Cells	Number Windows	Window Sizes as Fraction of max	Distribution of moves in windows	Final Cost	Percent Change
183	1	1	100%	76498	+0.0%
183	2	1 : 1/3	20% : 80%	67159	-12.2%
183	3	1 : 2/3 : 1/3	8% : 24% : 68%	69010	-9.8%
183	4	1 : ¾ : ½ : ¼	5% : 10% : 20% : 65%	650034	-10%
286	1	1	100%	115359	+0.0%
286	2	1 : 1/3	20% : 80%	102398	-11.2%
286	3	1 : 2/3 : 1/3	8% : 24% : 68%	102478	-11.2%
286	4	1 : ¾ : ½ : ¼	5% : 10% : 20% : 65%	98312	-14.8%